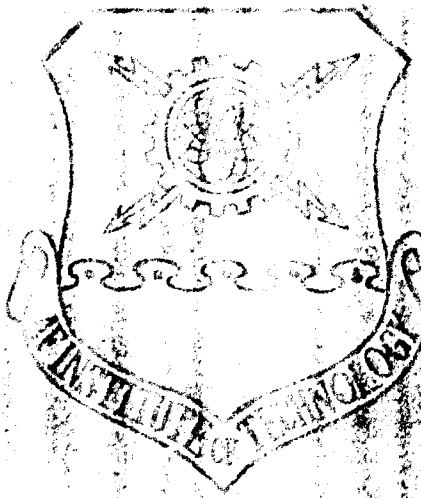


NETWORK REDUCTION USING ERROR PREDICTION

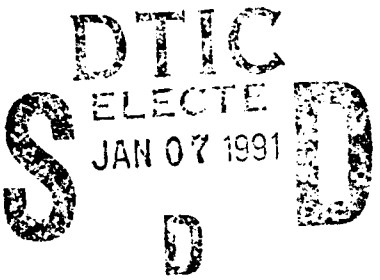
THESIS

Michael V. Gilsdorf, Captain, USAF

AFIT/GE/ENG/90D-24

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/90D-24



NETWORK REDUCTION USING ERROR PREDICTION

THESIS

Michael V. Gilsdorf, Captain, USAF

AFIT/GE/ENG/90D-24

AFIT/GE/ENG/90D-24

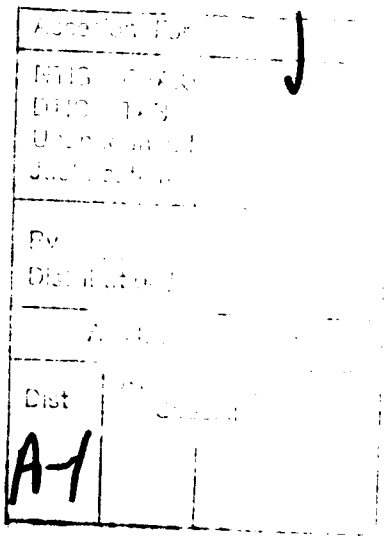
NETWORK REDUCTION USING ERROR PREDICTION

THESIS

Presented to the School Of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

Michael V. Gilsdorf, B.S.
Captain, USAF

December 1990



Acknowledgments

I would like to express my thanks to my family and friends who offered their prayers, words of encouragement, and support. In addition, I would like to express my special thanks to Dr. Steven K. Rogers, my thesis advisor, along with Dr. Mark E. Oxley for their assistance, special insight, and interesting discussions throughout this undertaking.

Table of Contents

Abstract

This thesis investigates gradient descent learning algorithms for multi-layer feed forward neural networks. A technique is developed which uses error prediction to reduce the number of weights/nodes in a network. The research begins by studying the first and second order back-prop training algorithms along with their convergence properties. A network is reduced by making an estimate of the amount of error which would occur when a weight(s) is removed. This error estimate is then used to determine if a particular weight is essential to the operation of the network. If not, it is removed and the network retrained. The process is repeated until the network is reduced to the desired size, or the error becomes unacceptable.

I Introduction

Background

In recent years neural networks have become a focus for study of many scholars because they hold the promise to solve complex problems in a rather short period of time. Some of today's problems can be difficult to solve, and may require many hours of computer time just to arrive at an approximate solution. Unfortunately, in many cases an answer may be needed in real time, (almost immediately) as soon as the input data is received or generated. Some examples which are getting a lot of attention today are pattern and speech recognition systems. These systems are being employed to operate or guide vehicles or defensive systems where a change initiated by the surrounding environment demands an immediate response. A defense system utilizing target recognition requires potential targets be identified quickly and accurately if it is to be of any use. Obviously, the accuracy and response time of a system are critical factors which can make major contributions to the final outcome. In situations where seconds count, it can make a difference between life and death - success or failure.

Mapping Functions

Scientist and mathematicians realized early on that mapping functions might hold the answer to solving complex problems rapidly. If the correct mapping function for a particular problem could be found, it could be employed to solve the problem relatively quickly and directly without the need of arduous time consuming repetitive calculations. The input data could simply be "presented" to the function, and the output solution calculated. However, finding the right mapping function for a given problem was not always an easy task.

Neural networks promised to overcome many of the difficulties associated with complex problems. Basically, a network can be thought of as a mathematical function which attempts to map a set of input variables to a desired set of output variables (i.e., classes). A network can provide a solution quite fast once it has been trained to respond to a particular problem. Training, on the other hand, can be very time intensive. It can range from minutes to hours or even days depending upon the accuracy desired. However, a network's response time is not necessarily hindered due to the training requirement. Depending upon the type of network, training can take place prior to the presentation of input (test) data. In this way a neural network can arrive at a solution quickly, simply by applying the test data to the mapping function.

Training

In some cases a network can fail its training process. This is generally caused by the failure of the training algorithm to converge to the proper set of parameters (weights). Reasons for this can vary, and can be attributed to such things as poor convergence properties of the training algorithm, incorrect initial conditions, poor network architecture, insufficient number of training examples, and/or poor input training data.

There are two factors which are major influences affecting training time. The first of these is the type of training algorithm used. Among training algorithms, two features which are highly desirable are small size (in terms of program code length), and good convergence properties. However, these two features tend to be self-opposing. In a general sense, improving the convergence property of an algorithm usually produces a larger size algorithm. For example, although second order training algorithms tend to converge faster, their algorithms are longer and require more time to compute than their first order counterparts. The key to choosing an algorithm with good training speed is to pick one which strikes an optimum balance between length of code and rate of convergence. Little is gained in terms of speed if the training algorithm converges twice as fast for each training iteration, but each iteration takes 10 times as long (Ruck, 1989).

The second major factor which affects training time is the size of the network or its architecture. Larger nets take longer to train due to the increase number of nodal calculations. Also, as the number of layers increase, the net's complexity increases too. Larger nets can, in certain instances, exhibit a higher degree of accuracy. However, increasing the size of the network does not in itself lead to better precision. Rather, accuracy is influenced by many factors among which is the accuracy and amount of training data. Certainly, garbage in will produce garbage out regardless of the size or architecture of the network.

Problem Statement

This research will investigate gradient descent learning algorithms for multi-layer feed forward networks. A technique will be developed that will use error prediction to determine the contribution of the various weights and nodes to the overall solution of a given size neural network. It will investigate ways to reduce the size of a net using both first and second order training algorithms and error prediction techniques.

Scope

The goal of this research is to devise a method that will train a network while at the same time determine which of the weights and nodes are essential and which are not. The non-essential weights are those weights which when removed result in the least amount of acceptable error. Equivalently, the same holds true for the nodes. A node can be removed if the error associated with removing all its weights results in an admissible error.

Current techniques which attempt to prioritize the importance of weights (or nodes) use a process called saliency. Saliency assign a number to each parameter (i.e., rank orders the weights or nodes) based upon its relative importance to the network. The saliency number for a particular parameter is based upon the amount of error induced at the output when that one parameter is removed from the net. Once the parameter with the lowest saliency is found, it is then permanently removed, the network retrained, and the process repeated again on the "pruned" network until the net is reduced to the desired size, or the output error is excessive. This entire training and pruning process can be slow and require a considerable amount of time especially for large networks. This research will attempt to improve upon this method by examining ways to improve the training algorithm (e.g., using a second order method), and by eliminating one or more parameters at the same time

based upon a prediction of the error. The amount of error associated with one or more weights will be estimated by using a Taylor series along with first and second order information gathered during the training process. By using this technique it is hoped that a network can be trained and reduced to its essential elements in a relatively short period of time.

For this research two layer networks will be used to study the training and reduction problem. A two layer network is defined as a network having one output layer and one hidden layer. One or both layers can employ non-linear squashing functions (e.g., sigmoids) at their nodal outputs. Two layer networks which have linear outputs are classified as Cybenko networks, and are capable of mapping any continuous function (Cybenko, 1989). These features make Cybenko networks attractive since they are less complex, and can be trained and reduced much faster than other multi-layer networks.

Methodology and Approach

This research will be divided into two principle phases. The purpose of the first phase is to examine the convergence properties of the first and second order back-propagation training algorithms and improve upon them. Before testing the algorithms on a neural network learning problem, however, they will first be applied to well understood functions (i.e., polynomials in one variable) to test their ability to converge to a known local minimum. The algorithms will be judged on how well they converge over a wide range of starting values including all critical points (points of inflection, minima, and maxima). In addition, the performance of the second order algorithm will be compared to the first order to study speed and convergence properties. One known drawback of the second order multi-variable algorithm is its requirement to invert a Hessian matrix. In general, the matrix may not be invertible, or may be too time consuming to invert repeatedly for every iteration. Should this be the case, either the first order algorithm will be temporarily re-employed, or an approximation will be made using only the elements lying along the principal or major diagonal (Le Cun, 1989). Once the algorithms are tested on the polynomial(s), they will be applied to various sizes of two layer networks to observe how well convergence takes place.

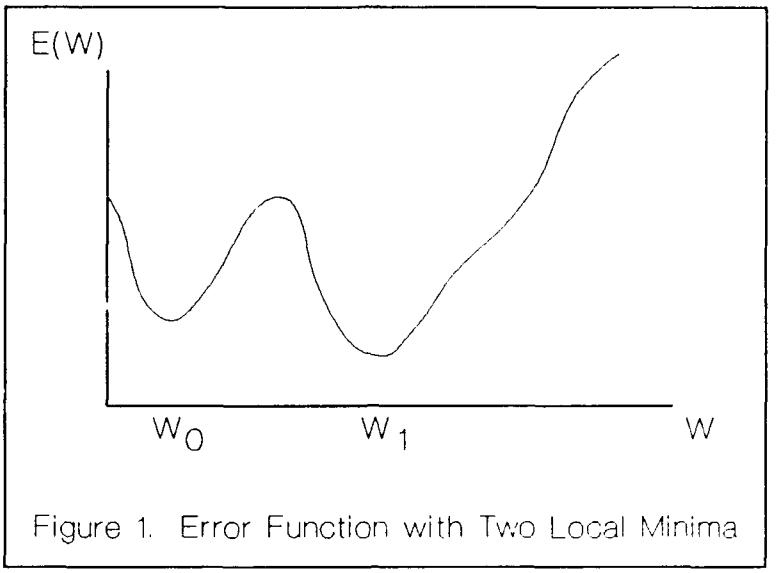


Figure 1. Error Function with Two Local Minima

The next phase will address the problem of error prediction and nodal reduction. As previously stated, a Taylor series along with the weight's first and second order information about the current value will be used to predict the amount of error when a weight is removed (i.e., set to zero). Since the higher order terms in the series expansion will not be computed, the Taylor expression is valid only for a small neighborhood about the current value. In other words, weights whose current value are already close to zero are the only candidates which will likely produce an accurate error prediction. See figure 1. The further away the weight is from zero, the more likely the Taylor series estimate will be in error due to the nature of the function and changes in its concavity. This limitation appears to severely restrict the number of weights which may be candidates for elimination. Should the training algorithm converge to a local minimum having weights

all with values much larger than zero, then the error estimate obtained with the Taylor series approximation will be highly in doubt. To counter this, careful attention will be devoted to ways of increasing the number of weights with values in the neighborhood of zero during the training process. Once the weights are chosen and their errors are calculated, the network will be pruned by eliminating those weights which have a very low relative error. Various size networks will be tested with the error prediction technique. Each will be trained to solve the XOR problem. This problem was chosen because it requires no more than three nodes - two hidden and one output to reach a solution (Rogers, 1990). Since the minimum network size for this problem is well known, it appears to be an excellent bench-mark for testing the effectiveness of the reduction technique.

The following recipe will be used to prune the network and determine its essential weights:

(1) Train the net using both first and/or second order algorithms while trying to maximize the number of weights in the neighborhood of zero.

(2) Using only those weights in the neighborhood of zero, make a prediction of the amount of error which would result when the weight is eliminated - set to zero. Make the error estimate using a Taylor series along with the first and second order information for the weight's current value.

(3) Retrain the net with the weights removed, and compare the error with the predicted error.

(4) Repeat steps (2) and (3) as necessary until the desired size net is achieved, or the error is unacceptable.

Thesis Organization

This thesis is divided into four chapters. The first chapter provides a brief introduction of the topic along with a problem statement and methodology. The literature review in chapter two consists of a presentation and review of material which has been published in the field relating to the problem at hand. Chapter three, the analysis and results, is comprised of a presentation and discussion of the equations and algorithms used along with the results of the research. The fourth chapter provides a conclusion and summary.

II Literature Review

Background

Neural networks have a unique ability to quickly solve complex problems ranging from object identification to speech recognition. They appear to offer a decisive advantage over other classical methods such as auto-correlation and Fourier analysis. These latter techniques can sometimes require many intensive and sometime repetitious calculations over vast amounts of data before reaching a solution. A trained neural net, on the other hand, can render a decision almost immediately when presented with the input data (i.e., feature set). In the case of pattern recognition, the feature set could include such characteristics as an object's size, shape, color, or aspect ratio.

One important but basic question which still remains largely unanswered pertains to the network architecture. How large should a network be for it to properly recognize an object, or make the correct decision? How many hidden layers and nodes are required? How should the hidden layers of a net be interconnected? Although rules of thumb and past experience have played key roles in network design in the past, a more disciplined method would be very desirable. For example, optimizing the size of a net could allow network engineers the means to design and build neural networks no

larger than is required to solve a particular problem. Not only can smaller nets be more economical to design and construct, but they can provide real benefits in terms of their operating ability. Smaller nets not only learn faster, but require less training examples. Generalization is another factor which can be influenced by the network architecture. Generalization is a term used to describe a network's ability to correctly respond to new input data which it has not been trained on. That is, the new data is not contained in the training data. Le Cun showed that networks which contain too many nodes and interconnections tend to generalize rather poorly (Le Cun, 1990).

Explanation and Description

Neural networks can be quite large, consisting of many layers of interconnecting nodal neurons. These networks can be constructed via software or hardware methods; however, software methods offer greater flexibility at the design level making changes much easier to implement. Figure 2 shows a typical feed forward network comprised of nine nodes and their interconnections. These nodes are arranged in three layers with the bottom and top layers providing the inputs and outputs of the network, respectively. Since only the top two layers have summing nodes, the net is referred to as a two-layer network. The bottom layer nodes serve only as inputs to the network. The middle layer is called a "hidden" layer since it is sandwiched between two other layers and its outputs drive the inputs of the layer immediately above it.

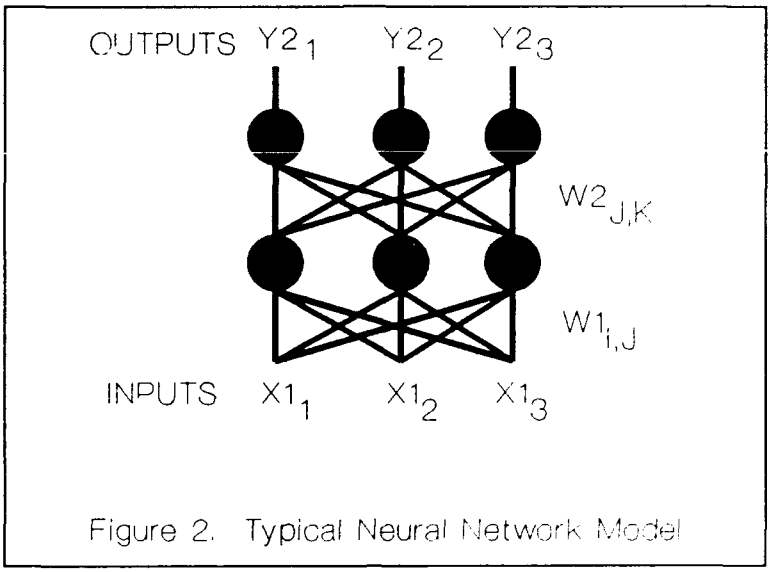


Figure 2. Typical Neural Network Model

Notice that each layer has multiple input and output connections. For each layer, input connections are situated at the bottom of the node; whereas, the output connections are located at the top. Nets of this type are referred to as multi-layer perceptrons. Also notice that each interconnection has a weighting factor (i.e., weight) associated with it. See figure 3. The weight is a numeric value assigned to each interconnection which determines the influence or gain of a particular connection. One input connection is assigned an input value of one, and has a weight associated with it known as the bias. This combination provides the summing node an offset allowing its output to vary as needed independent of the other inputs. It is analogous to the DC component in a Fourier transform.

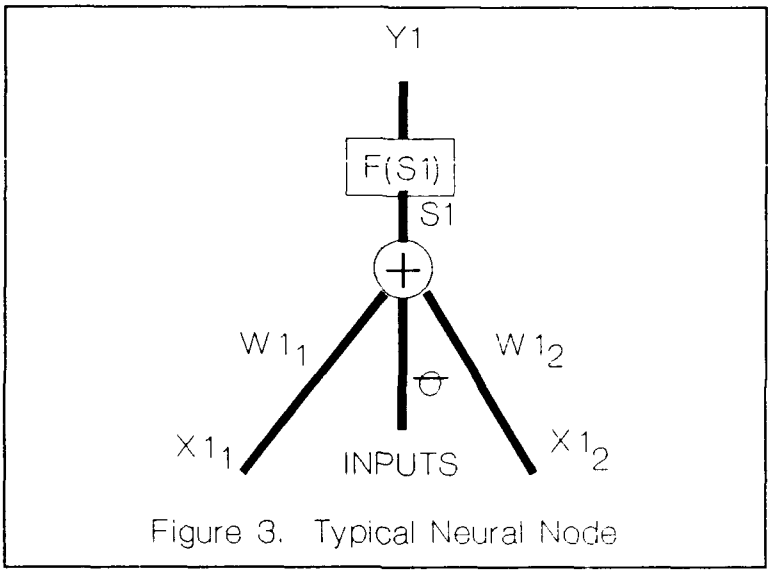


Figure 3. Typical Neural Node

The output nodes of a network relate directly to the number of classes the net is asked to distinguish. For example, if a net were designed to recognize one object from a set of 10, the net could be designed with 10 outputs - one per object. Typically, a network recognizes an object by setting the output corresponding to the found object to a value of one, while setting all other outputs to zero.

In order for a network to correctly respond to a given problem, it must first be trained. Training is the process by which the weights of a network are calculated and assigned values. It is their values which make a net respond to a specific input stimuli, and allow it to map the input data to the desired output classification. The values assigned are unique and relate directly to the task at hand. However, determining the values for

the weights is a non-trivial problem. Various different training algorithms have been formulated with the Back Propagation (or error correction) algorithm being the most widely used (Werbos, 1974). This training algorithm requires one to apply the known input features to the network, and then perform an iterative computation to determine the network's weights. This process begins by initializing the weights (e.g., some random number), then have the net calculate the outputs. The calculated outputs are compared to the desired outputs, and any differences or errors are used to generate a new set of values for the weights. The process is then repeated using the new weights, and continues until the desired output error is achieved. Once the iterative process is complete, the network has learned which weights to use for the given problem.

Justification

The reason for this literature review is to present a brief overview of the work conducted by other researchers which may contribute to solving the network reduction problem. Also, to understand the problem, one must first have knowledge of the current research in this field. Certainly, it would be helpful to examine reduction techniques applied to other areas of the network (e.g., feature input space), and determine whether the same techniques may be applied to reduce the number of nodes and

weights. One critical question which will be explored is what happens to a network when its nodes are removed. Can it still continue to operate effectively? If not, what performance factors are degraded, and in what way? How will the net's ability to generalize be affected? Another area of concern deals with the training algorithms used to determine the weights of the network. Since each node is fed by a set of weights, a node might not be needed if all weights feeding the node have very small values approaching zero. By examining the training algorithms, it is hoped that the network could be pruned while it is being trained (i.e., reduced on the fly).

Scope

This literature review will focus on areas which concern themselves directly with neural networks. Also, since the problem is to reduce the size of the network, it is important to examine any network component or factor which directly or indirectly affects the quantity or behavior of the weights and nodes. This includes both input and output parameters, as well as training algorithms. Each is hoped will lead to solving a bit of the puzzle.

Methodology and Organization

This literature review will begin by presenting information which describes how a network operates, and how the number of nodes affect the net's recognition process. It will also include a discussion of factors which are known to determine the number of input and output nodes. After addressing network generalities, an examination of the some of the more popular training algorithms will be presented. This discussion will look at some of the current techniques used to generate values for the weights in the network.

Discussion of the Literature

Designing and constructing smaller size neural networks lends advantages in reduced construction costs as well as training time. How large a network should be to solve a given problem is only partially understood. Sizing the number of nodes for the input and output layers is known to depend upon the I/O environment. For example, the number of input nodes relate directly to the number of input features. The more input features needed to distinguish a particular class or object, the more input nodes are required. The problem of determining which features are essential and which are not was studied by Ruck. He presented a method which was able to distinguish between essential and non-essential features (Ruck,

1989). As a result non-essential features could be discarded, which in turn eliminated input nodes and reduced the size of the net's input layer.

The number of output nodes of a network is dependent upon the number of classifications (e.g., objects) the net is asked to recognize. Each classification requires one output node. As in the case of the input nodes, the output nodes have a one-to-one relationship.

Little is known about how to determine the number of hidden layers, or for that matter, the number of nodes in the layer. Presently, many engineers and designers use rules of thumb, or trial and error methods. Although nets designed in this manner can operate quite acceptably, they can contain weights and nodes which appear to be non-essential. Tarr was able to demonstrate this, and show that some nets contained nodes within the hidden layers which appeared to contribute little to the net's problem solving ability (Tarr, 1988). He also discovered that as certain inner nodes were removed from the network, the net could still continue to perform satisfactorily.

Various algorithms have been formulated to train networks. Each have their advantages and disadvantages. Some are faster while others not as precise. Still others fail to converge to the

set of weights which provide the minimum output error. Much of the research on neural networks is focusing on these problems. Chiueh's paper is an excellent source for training algorithms (Chiueh, 1988). He discusses many of the more common algorithms used for training neural nets. Although the Back Propagation algorithm is perhaps the most popular training algorithm due to its simplicity, it is by no means perfect. Not only may it require many thousands of iterations to train a net, it sometimes fails to converge to the correct set of weights. Becker showed a way to improve the convergence properties of the back-prop algorithm. Her technique involved the use of second order methods which promised a reduction in the number of iterations with only a small increase in computational complexity and training time (Becker, 1989).

Singhal and Wu decided to apply the Extended Kalman filtering algorithm and test its training ability on multi-layer perceptrons (Singhal, 1989). The Kalman algorithm works by approximating a linear region around the current estimate within the non-linear field. This linear region is updated as new estimates are calculated until the change (or error) is very small. Singhal and Wu tested the algorithm on two classic problems, the XOR and Meshed Disconnected regions. The results were then compared to those obtained using the back propagation algorithm. The number of iterations for both algorithms was limited to a maximum of

approximately 2000, with the error calculated using the root mean square error over all the training data.

Singhal and Wu claimed the Extended Kalman algorithm trained in just a "few iterations". Their results showed that the Kalman algorithm achieved a lower error after 5 or 10 iterations than the back-prop achieved after 2000. However, one must be careful in drawing any overall conclusion from this. First, the Kalman algorithm was tested and compared on only two sets of training data, and the size of the net used was relatively small compared to those used to solve more complex problems such as speech recognition. How well a complex net would train using an Kalman algorithm is yet to be seen. Second, although the Kalman trained in fewer iterations, the number of calculations required for each iteration was much more intensive than that required by the back-prop. Therefore, when comparing training times, the advantage of the Kalman algorithm diminishes (Ruck, 1990).

It is not fully understood why a given training algorithm sometimes fails to converge; however, convergence can sometimes be achieved by changing the number of input parameters or the size of the net. Also, an algorithm may fail to converge if the initial values assigned to the weights were inappropriate. Convergence can sometimes be achieved by assigning a new set of weights to the net and running the algorithm again. Wang has studied this problem and

proposed a training algorithm which exhibits strong global convergence properties for varying network sizes and configurations (Wang, 1988). His algorithm uses both first and second order derivatives to ensure convergence. However, sometimes an algorithm can converge to a local minimum rather than a global minimum. Baba suggests that perhaps the Matyas and its modified algorithm might be applied to the back-prop error function to ensure a global minimum (Baba, 1989). But again, these techniques appear to be at the expense higher complexity and longer training times.

Ahmad and Tesauro were attempting to gain some insight into the relationships between the size of the net, the number of training patterns, and their effect upon the net's generalization ability for various changes to the inputs (Ahmad, 1989). Their results showed that for a given network size, its ability to correctly respond to a set of inputs was proportional to the number of training sets. They found that the failure rate decreased exponentially as the number of training sets applied to the net increased. Ahmad and Tesauro also found that the failure rate was approximately constant for a given number of training sets even though the number of inputs varied. This may have been due to the somewhat boolean-like linear function the net was being trained to respond to. Whether the relationship is true in general is unknown. Also, they reported that a "minor" change to the input produced a significant change in the net's performance. It was

also stated that the specific training patterns used had a large influence on the net's final weights and its ability to generalize. Although their work was not conclusive, it did provide some addition insight into the relationships which affect training. Their mathematical analysis presented may be a basis for further research.

Some researchers have already begun looking at ways to reduce or optimize the size of neural networks. Le Cun, Denker, and Solla published a paper which examined the use of a second order technique to reduce the number of nodes and weights in a network (Le Cun, 1990). They presented a "recipe" which describes how to perform the reduction. Their results appeared to show that the second order technique to be quite effective - at least for the given problem (recognizing hand-written characters). However, the results may not be true in general. The method they used to calculate the inverse matrix was an approximation which ignored the off diagonal terms. This approximation is based upon three assumptions, of which the failure of any one, invalidates the approximation. Although the general methodology presented appears valid, further study may be needed before the technique should be applied to nets in general.

Sietsma also studied ways to prune nets and their effect on generalization. She states:

> The ability to generalize, to give correct outputs for inputs not in the training set, is probably the most important strength of artificial neural networks. At the same time "good generalization" is difficult to define. ... If the training set does not adequately define the classes, the networks may generalize in a way which is very "reasonable" but which is not what was intended or desired by the experimenter (Sietsma, 1990).

She continues by saying that she found that nets which were pruned (or minimized) are frequently plagued by local minima when undergoing training. It was unclear what she meant by this since the training process is meant to converge to a local minimum. Supposedly, she was referring to the situation where the training algorithm converged (or appeared to converge) to a local minimum where the error was unacceptable.

Ishikawa took a different approach to the minimization problem. He suggests the problem of minimization can be best overcome by the use of a new training algorithm, a structured learning algorithm. He states that although the back-prop algorithm is simple and popular, it suffers from two serious drawbacks. First, it requires the designer to draw up an initial model of a network prior to reduction. This model is generally developed by trial and error methods, or rules of thumb based upon past experience. The designer almost needs some prior knowledge of the network - something which is not available in many instances.

The second problem which Ishikawa discusses is the difficulty one has at interpreting the meaning of the nodes in the inner (or hidden) layers. Since the network is modeled in a non-structured manner, the nodes in the inner layer have a meaning to the net, but may have no direct interpretation to humans (Ishikawa, 1989).

Conclusion

The problem of determining the number of nodes necessary for a network to solve a given problem appears to depend significantly on its initial architecture and training algorithm. Portions of the architecture consisting of the input and output layers can be analyzed to determine the required number of nodes. The number of input nodes depends on the number of input features representing the input data. Reducing the object's feature set can reduce the number of input nodes. On the other hand, the number of output nodes depends upon the number of objects or items the network is asked to classify. When asked to differentiate and recognize a large number of classifications, both the network and the feature set can be quite large. However, determining the number of nodes lying in the inner layers of the net is for the most part an unsolved problem. An analysis of the training algorithms may hold the key to understanding how to determine which inner nodes are essential. The training algorithms determine the weights, and the weight's value determine the importance of a node to the network.

III Analysis and Results

Network Equations

Since a Taylor series will be used to make an estimate of the error when a weight is nulled, we begin by deriving the equation which expresses the error as a function of the weights. This is done by generating the equations at each layer, beginning at the top, and continuing down to the lowest layer in the net. Although this derivation is for a 2-layer network, the process is the same for a network composed of any number of layers. We begin by defining the total network error over all training samples to be:

(1)

$$E_T(\vec{W}) = \sum_{p=1}^{P} E_p$$

where

E_T is the total error
E_P is the input pattern error
$\vec{W}$ are the weights.

Now, the pattern error is the difference between the net's calculated output and the desired output for a specific input pattern. Notice, however, that if the pattern error is always greater than or equal to zero, then the total error can be minimized simply by minimizing each pattern error. To ensure the pattern error is non-negative, we define the error for the **p**th input pattern to be:

(2)

$$E_p = \sum_{k=1}^{K} \frac{1}{2}(D_k - Y2_k)^2$$

where

D_k is the desired output
$Y2_k$ is the 2nd layer output
K is the number of 2nd layer outputs.

Defining the pattern error in this manner provides another advantage which will be exploited later, namely the derivative is simpler.

Working down from the top of the net, the equations at each layer can be derived. For a particular **k**th output the expression is:

$$Y2_k = f_k(S2_k) \tag{3}$$

where

$S2_k$ *is the 2nd layer summation output*
f_k *is the 2nd layer function*
Linear:
$f_k(S2_k) = S2_k$
$f_k'(S2_k) = 1$
$f_k''(S2_k) = 0$
Sigmoid:
$f_k(S2_k) = (1 + e^{-S2_k})^{-1}$
$f_k'(S2_k) = f_k^2(S2_k) - f_k(S2_k)$
$f_k''(S2_k) = f_k'(S2_k) - 2\, f_k'(S2_k) f_k(S2_k)$.

For a Cybenko network, the linear function is used in lieu of the sigmoid (Cybenko, 1989).

Next, the equation which expresses the output of the **k**th summing node in terms of its inputs is:

(4)

$$S2_k = \sum_{j=1}^{J} W2_{j,k} Y1_j$$

where

$W2_{j,k}$ *is the 2nd layer weight*
$Y1_j$ *is the 1st layer output*
J *is the number of 1st layer outputs.*

Now, the ingredients are on hand to express the pattern error as a function of the 2nd-layer weights. Combining the last three expressions gives:

(5)

$$E_p = \sum_{k=1}^{K} \frac{1}{2}\left(D_k - f_k\left(\sum_{j=1}^{J} W2_{j,k} Y1_j\right)\right)^2 \quad .$$

Next, the partial derivatives for the error pattern with respect to a 2nd-layer weight are computed. The first partial derivatives can be derived by using the chain rule:

(7)

$$\frac{\partial E_p}{\partial W2_{j,k}} = \sum_{\tilde{k}=1}^{K} \frac{\partial E_p}{\partial Y2_{\tilde{k}}} \frac{dY2_{\tilde{k}}}{dS2_{\tilde{k}}} \frac{\partial S2_{\tilde{k}}}{\partial W2_{j,k}}$$

Since

$$\frac{\partial S2_{\tilde{k}}}{\partial W2_{j,k}} = 0 \quad for\ \tilde{k} \neq k$$

then (8)

$$\frac{\partial E_p}{\partial W2_{j,k}} = \frac{\partial E_p}{\partial Y2_k} \frac{dY2_k}{dS2_k} \frac{\partial S2_k}{\partial W2_{j,k}}$$

where

$$\frac{\partial E_p}{\partial Y2_k} = Y2_k - D_k$$

$$\frac{dY2_k}{dS2_k} = f_k'(S2_k)$$

$$\frac{\partial S2_k}{\partial W2_{j,k}} = Y1_j.$$

Also, the second partial derivatives for the 2nd-layer can be derived using the chain rule along with product rule:

(9)

$$\frac{\partial^2 E_p}{\partial W2_{j,k}^2} = \frac{\partial^2 E_p}{\partial W2_{j,k}\partial Y2_k}\frac{dY2_k}{dS2_k}\frac{\partial S2_k}{\partial W2_{j,k}} + \frac{\partial E_p}{\partial Y2_k}\frac{\partial^2 Y2_k}{\partial W2_{j,k}\partial S2_k}\frac{\partial S2_k}{\partial W2_{j,k}} + \frac{\partial E_p}{\partial Y2_k}\frac{dY2_k}{dS2_k}\frac{\partial^2 S2_k}{\partial W2_{j,k}^2}$$

where

$$\frac{\partial^2 S2_k}{\partial W2_{j,k}^2} = 0$$

$$\frac{\partial^2 Y2_k}{\partial W2_{j,k}\partial S2_k} = \frac{d^2 Y2_k}{dS2_k^2}\frac{\partial S2_k}{\partial W2_{j,k}} = f_k''(S2_k)\,Y1_j$$

$$\frac{\partial^2 E_p}{\partial W2_{j,k}\partial Y2_k} = \frac{\partial^2 E_p}{\partial Y2_k^2}\frac{dY2_k}{dS2_k}\frac{\partial S2_k}{\partial W2_{j,k}} = f_k'(S2_k)\,Y1_j.$$

After substituting and simplifying, the partial derivative equations for the 2nd-layer finally become:

(10)

$$\frac{\partial E_p}{\partial W2_{j,k}} = (Y2_k - D_k)\; f_k'(S2_k)\; Y1_j$$

and

(11)

$$\frac{\partial^2 E_p}{\partial W2_{j,k}^2} = Y1_j^2\; f_k'(S2_k)^2 + Y1_j^2\; f_k''(S2_k)\;(Y2_k - D_k).$$

Deriving the error expressions for the bottom layer is performed much like it was for the top. The equation expressing the jth output of the 1st-layer is:

$$Y1_j = f_j(S1_j) \tag{12}$$

where

$S1_j$ *is the* 1st *layer summation output activation*
f_j *is the* 1st *layer squashing function*
Sigmoid:

$$f_j(S1_j) = \left(1 + e^{-S1_j}\right)^{-1}$$
$$f_j'(S1_j) = f_j(S1_j) - f_j^2(S1_j)$$
$$f_j''(S1_j) = f_j'(S1_j) - 2\, f_j'(S1_j) f_j(S1_j).$$

The equation which expresses the output of the jth summing node in terms of its inputs is:

$$S1_j = \sum_{i=1}^{I} W1_{i,j} X1_i \tag{13}$$

where

$W1_{i,j}$ *is the* 1st *layer weight*
$X1_i$ *is the input*
I *is the number of inputs.*

The final equation which expresses the pattern error as a function of both the weights and the input can now be formulated. It is found by back substituting the last two expressions into the 1st-layer equation.

(14)

$$E_p = \sum_{k=1}^{K} \frac{1}{2}\left(D_k - f_k\left(\sum_{j=1}^{J} W2_{j,k} f_j\left(\sum_{i=1}^{I} W1_{i,j} X1_i\right)\right)\right)^2$$

The partial derivatives for the pattern error with respect to a 1st-layer weight can now be computed using the chain rule. The first partial derivative becomes:

(15)

$$\frac{\partial E_p}{\partial W1_{i,j}} = \sum_{k=1}^{K} \sum_{\tilde{j}=1}^{J} \frac{\partial E_p}{\partial Y2_k} \frac{dY2_k}{dS2_k} \frac{\partial S2_k}{\partial Y1_{\tilde{j}}} \frac{dY1_{\tilde{j}}}{dS1_{\tilde{j}}} \frac{\partial S1_{\tilde{j}}}{\partial W1_{i,j}}$$

Since

$$\frac{\partial S1_{\tilde{j}}}{\partial W1_{i,j}} = 0 \quad for\ \tilde{j} \neq j$$

then

(16)

$$\frac{\partial E_p}{\partial W1_{i,j}} = \sum_{k=1}^{K} \frac{\partial E_p}{\partial Y2_k} \frac{dY2_k}{dS2_k} \frac{\partial S2_k}{\partial Y1_j} \frac{dY1_j}{dS1_j} \frac{\partial S1_j}{\partial W1_{i,j}}$$

where

$$\frac{\partial S2_k}{\partial Y1_j} = W2_{j,k}$$

$$\frac{dY1_j}{dS1_j} = f_j'(S1_j)$$

$$\frac{\partial S1_j}{\partial W1_{i,j}} = X1_i.$$

The remaining partial derivatives were previously computed.

The second partial derivatives for the 1st-layer can also be derived. (17)

$$\frac{\partial^2 E_p}{\partial W1_{i,j}^2} = \sum_{k=1}^{K} \frac{\partial^2 E_p}{\partial W1_{i,j}\partial Y2_k} P_2 P_3 P_4 P_5 + P_1 \frac{\partial^2 Y2_k}{\partial W1_{i,j}\partial S2_k} P_3 P_4 P_5$$

$$+ P_1 P_2 \frac{\partial^2 S2_k}{\partial W1_{i,j}\partial Y1_j} P_4 P_5 + P_1 P_2 P_3 \frac{\partial^2 Y1_j}{\partial W1_{i,j}\partial S1_j} P_5$$

$$+ P_1 P_2 P_3 P_4 \frac{\partial^2 S1_j}{\partial W1_{i,j}^2}$$

where:

$$\frac{\partial^2 S1_j}{\partial W1_{i,j}^2} = 0$$

$$\frac{\partial^2 Y1_j}{\partial W1_{i,j}\partial S1_j} = \frac{d^2 Y1_j}{dS1_j^2}\frac{\partial S1_j}{\partial W1_{i,j}} = f_j''(S1_j)\, X1_i$$

$$\frac{\partial^2 S2_k}{\partial W1_{i,j}\partial Y1_j} = 0$$

$$\frac{\partial^2 Y2_k}{\partial W1_{i,j}\partial S2_k} = \frac{d^2 Y2_k}{dS2_k^2}\frac{\partial S2_k}{Y1_j}\frac{dY1_j}{dS1_j}\frac{\partial S1_j}{\partial W1_{i,j}} = f_k''(S2_k)\, W2_{j,k} f_j'(S1_j)\, X1_i$$

$$\frac{\partial^2 E_p}{\partial W1_{i,j}\partial Y2_k} = \frac{\partial^2 E_p}{\partial Y2_k^2}\frac{dY2_k}{dS2_k}\frac{\partial S2_k}{\partial Y1_j}\frac{dY1_j}{dS1_j}\frac{\partial S1_j}{\partial W1_{i,j}} = f_k'(S2_k)\, W2_{j,k} f_j'(S1_j)\, X1_i$$

$$P_1 = \frac{\partial E_p}{\partial Y2_k} = Y2_k - D_k$$

$$P_2 = \frac{dY2_k}{dS2_k} = f_k'(S2_k)$$

$$P_3 = \frac{\partial S2_k}{\partial Y1_j} = W2_{j,k}$$

$$P_4 = \frac{dY1_j}{dS1_j} = f_j'(S1_j)$$

$$P_5 = \frac{\partial S1_j}{\partial W1_{i,j}} = X1_i \quad .$$

After substituting and simplifying the above expressions, the partial derivative equations for the 1st-layer finally simplify to:

(18)

$$\frac{\partial E_p}{\partial W1_{i,j}} = \sum_{k=1}^{K} (Y2_k - D_k)\, f_k'(S2_k)\, W2_{j,k}\, f_j'(S1_j)\, X1_i$$

and

(19)

$$\frac{\partial^2 E_p}{\partial W1_{i,j}^2} = \sum_{k=1}^{K} f_k'(S2_k)^2\, f_j'(S1_j)^2\, W2_{j,k}^2\, X1_i^2 + (Y2_k - D_k)\, f_k''(S2_k)\, f_j'(S1_j)^2\, W2_{j,k}^2\, X1_i^2 + (Y2_k - D_k)\, f_k'(S2_k)\, f_j''(S1_j)\, W2_{j,k}\, X1_i^2 \quad .$$

Taylor Series

This research will use the Taylor series in two important areas: (1) the training algorithm and (2) the error predicting function. The Taylor series for a multi-variable expression is defined as:

(20)

$$E_p(\vec{W}) = E_p(\vec{W}_0) + (\vec{W}-\vec{W}_0)^T \nabla E_p(\vec{W}_0) + \frac{1}{2} (\vec{W}-\vec{W}_0)^T \nabla^2 E_p(\vec{W}_0) (\vec{W}-\vec{W}_0) + \cdots$$

where

$\vec{W}$ *are the weights* $W2_{j,k}$; $W1_{i,j}$

$\vec{W}_0$ *is the current value of* $\vec{W}$

$\nabla E_p(\vec{W}_0)$ *is the gradient of* E_p *evaluated at* $\vec{W}_0$

$\nabla^2 E_p(\vec{W}_0)$ *is the Hessian of* E_p *evaluated at* $\vec{W}_0$

To train a network, an algorithm is chosen which will find the set of weights which will minimize the error. First impressions might lead one to believe that finding a root of the error function will provide the correct set of weights. However, notice that the error expression is never negative, and additionally, there is no guarantee the error function will ever be zero for any set of weights. Therefore, a better algorithm to use is one that searches for a local minimum. Essentially, this process can be viewed as an unconstrained minimization problem.

Most algorithms which find a root of a function can be used to search for a local minimum (or maximum). Elementary calculus tells us that for a differentiable function a local minimum must occur where the first derivative of a function is zero. Hence, searching for a root of the first derivative will either provide a minimum or maximum. Newton's method is one such algorithm which can be used to search for a root. It can be easily altered to search for a minimum or maximum. The algorithm has its origins with the Taylor series, and uses only the first few terms of the expansion to approximate a function near a point. By applying the algorithm iteratively, it may or may not converge depending upon the nature of the function and initial conditions.

Using the Taylor series to approximate a multi-variable function (e.g., error function), Newton's second order algorithm for finding a minimum or maximum is derived as follows:

$$E_p(\vec{W}_1) = E_p(\vec{W}_0) + \vec{S}^T \nabla E_p(\vec{W}_0) + \frac{1}{2}\vec{S}^T \nabla^2 E_p(\vec{W}_0)\vec{S} \tag{21}$$

Since

$$\nabla E_p(\vec{W}_1) = \vec{0} \text{ at a maximum or minimum point,}$$

then

$$\vec{0} = \nabla E_p(\vec{W}_0) + \nabla^2 E_p(\vec{W}_0)\vec{S}$$

or

$$\vec{S} = -[\nabla^2 E_p(\vec{W}_0)]^{-1} \nabla E_p(\vec{W}_0) \qquad \text{where } \vec{S} = \vec{W}_1 - \vec{W}_0 \tag{22}$$

.

Notice, this algorithm requires the inverse of the Hessian matrix - a time consuming task. Furthermore, there is no guarantee the matrix will be invertible. Le Cun's method uses a slightly different form of the algorithm. It ignores the off diagonal elements of the Hessian, and treats it as a diagonalized matrix. Hence, the inversion is much simpler and faster (Le Cun, 1989). Using this Hessian approximation, the algorithm can now be expressed in terms of one weight along with its first and second order information:

(23)

$$W_1 = W_0 - \frac{\frac{\partial E_p}{\partial W}(W_0)}{\frac{\partial^2 E_p}{\partial W^2}(W_0)} \quad .$$

This algorithm provides the foundation for the first and second order back-prop training algorithm. With a slight modification it can be used to search for a minimum by forcing the algorithm to "follow" the direction of the negative gradient.

This algorithm (24) is referred to as a pseudo second order back-prop, hereby referred to as simply the second order training algorithm:

(24)

$$W_1 = W_0 \frac{\frac{\partial E_p}{\partial W}(W_0)}{\left|\frac{\partial^2 E_p}{\partial W^2}(W_0)\right|} .$$

The first order back-prop is merely a degenerate form of this. It still uses the first order term, but instead utilizes a constant (eta inverse) to approximate the second order term:

(25)

$$W_1 = W_0 - \eta \frac{\partial E_p}{\partial W}(W_0) .$$

Convergence Analysis

In order to achieve the stated goal of network minimization through error prediction, it was necessary to examine the back-prop training algorithm(s) along with their convergence properties. This was essential if the number of weights with final values close to zero were to be maximized. After all, it is the training algorithm which determines the final values for the weights as it searches to minimize the error function. Should the algorithm have poor convergence properties in the vicinity of zero, it may diverge and/or converge to an entirely different set of weights far removed from zero, thus presenting less weights as candidates for error prediction.

Recall from the earlier discussion, that a Taylor series truncated after just a few terms can only approximate a function. The series is most accurate in the neighborhood of the current weight value. Attempting to interpolate beyond this neighborhood increases the chance of error, and can give an incorrect evaluation of the function. Applying this fact to the problem at hand, namely predicting the error when a weight is set to zero (i.e., nulled), presents a major problem. Suppose a network is trained (error is minimized), and the local minimum is such that all weights have values much greater than zero. This would result in a situation where the Taylor series would not give an accurate estimate of the error when a weight is nulled. It is certainly possible that there may exist another local minimum which does have one or more weights with values close to zero. But, how do we find it? How do we minimize the error function while at the same time maximizing the number of weights in the neighborhood of zero?

The first attempt at maximizing the number of weights in the vicinity of zero revolved around the question of whether initializing the weights to values close to zero would increase the number of weights with final values close to zero. The answer seemed to be a qualified yes, provided the value(s) were within an interval of convergence. However, should this not be the case (e.g., close to an inflection point), then the algorithm could diverge to another set of weights far removed. To test this

hypothesis, a two-layer network (15 hidden nodes) was trained to solve the XOR problem using the first order training algorithm. First, it was trained using initial values uniformly distributed between -5.0 to 5.0, -3.0 and 3.0, -1.0 and 1.0, and then finally between -0.5 and 0.5. After the network was trained the weights were examined to determine how many had final values between -1.0 and 1.0. The results are tabulated below.

Table I. Weight Value and Count

Initial Weights of Weights	Number of Weights Between -1.0 and 1.0
-5.0 to 5.0	5
-3.0 to 3.0	6
-1.0 to 1.0	8
-0.5 to 0.5	7

The results were inconclusive, but they did appear to show that when the weights were initialized with values close to zero, the chances of having weights with final values close to zero were increased.

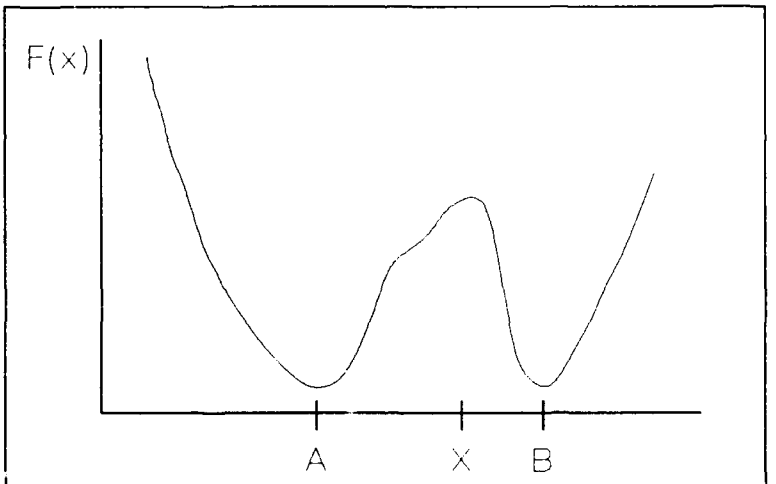


Figure 4. Pt A is "Closest Minimum" to Pt X

In hopes of further increasing the number of weights close to zero, the convergence properties of the training algorithm(s) were examined. For this analysis it was necessary to apply the algorithms to a function with known points of inflection, minima, and maxima to test how each algorithm was able to converge to the "closest minimum". The "closest minimum" to a point **x** is defined as the local minimum where the function is monotonic for the interval bounded by a local minimum and point **x**. See Figure 4. In other words, even though point **B** is closer to point **x**, the algorithm will have a tendency to converge towards point **A** due to the monotonic behavior of the function between points **A** and **x**.

The function chosen to study the convergence properties of the training algorithms was a 6th degree polynomial in **w**, a weight.

$$p(w) = (w-1)(w-2)^2(w-3)(w-4)(w-5) + 10 \tag{26}$$

Like the error function, the polynomial was positive for all values of **w**. Although a polynomial in one variable does not approximate a network error function, it can, nevertheless, be useful for testing the convergence ability of an algorithm. Certainly, if the algorithm exhibits poor convergence tendencies for the polynomial, its ability to converge for a network error function will be questionable.

The first algorithm tested was the first order back-prop. With a small value of eta (0.035) the algorithm converged, but its convergence was very slow (27 iterations), especially when the initial value was far from the local minimum. When the algorithm eventually converged, it did not always converge to the "closest minimum". When eta was increased to 0.35 the algorithm approached a local minimum much quicker, but failed to converge. Instead it had a tendency to consistently overshoot the minimum - bouncing back and forth. The results of this test showed that the value of eta had a marked effect on the ability of the first order algorithm to converge to a local minimum.

It appeared that a large value of eta was necessary when far from the local minimum, while a small value of eta was necessary when close to the minimum. To capitalize on this finding, it was decided to rewrite the first order algorithm to allow for a varying eta. Eta was initialized to a value of 0.35, and its value was doubled if the sign of the first derivative was unchanged. If the sign changed, indicating the presence of a local minimum, the value of eta was reduced by 3/8ths. After making this modification, the first order algorithm was retested. It was found to converge to a local minimum much faster than before, but it was not always to the "closest minimum. With the same initial value as before (50), the algorithm converged in about half the number of iterations (12 versus 27). These results varied depending upon the initial value, but in all 10 cases using different initial conditions the convergence rate was equal or better with the varying eta than with a fixed eta.

Next, the first order algorithm with a momentum term was applied to the polynomial.

(27)

$$W_2 = W_1 - \eta \frac{\partial E_p}{\partial W_1} + \alpha (W_1 - W_0)$$

where

W_0 *is the previous weight*
W_1 *is the current weight*
W_2 *is the updated weight*
α *is a fixed constant.*

Again, using a fixed eta (first 0.035 and then 0.35) and an alpha of 0.7, the algorithm failed to converge, but consistently overshot the local minimum. When compared to the fixed eta algorithm without the momentum term, it did approach the region of the local minimum in less iterations. However, the results did not fair as well as those using the varying eta. One important fact was observed with all 3 first order algorithms. If the value of the first derivative was at or near zero, the algorithm either failed to converge to a local minimum, or converged very slowly. None of the first order algorithms consistently converged to the "closest minimum"; therefore, they did not offer much promise in solving the network reduction problem.

Next, the second order back-prop was tested using the same polynomial. Recall the algorithm is:

(28)

$$W_1 = W_0 - \frac{\frac{\partial E_p}{\partial W}(W_0)}{\left|\frac{\partial^2 E_p}{\partial W^2}(W_0)\right|} \quad .$$

Notice, the algorithm could fail to converge to a minimum should its initial value place it at a local maximum (i.e., first derivative is zero). Also, notice that the algorithm is undefined at a point of inflection (i.e., second derivative is zero). To prevent these events from occurring, a check was performed to ensure the algorithm was not at a maximum point, and the algorithm was modified slightly as follows:

(29)

$$W_1 = W_0 - \frac{\frac{\partial E_p}{\partial W}(W_0)}{\left|\frac{\partial^2 E_p}{\partial W^2}(W_0)\right| + \epsilon}$$

where $\epsilon << desired\ error$.

When the algorithm was tested on the polynomial, the results showed it to "always" converge to a local minimum regardless of the initial value. When the initial value was on or near a point of inflection, the algorithm provided a sequence of weights that moved to some distant point, but then eventually found its way to a local

minimum. However, this tendency to "always" converge may be due to the nature of the polynomial rather than a property of the algorithm itself. At the extremities of the given polynomial, there were no inflection points between an extreme value of **w** and the "closest minimum". Hence, once the algorithm diverged to a distant point, the algorithm was able to converge to the "closest minimum" without passing through a point of inflection.

The ability of the second order algorithm to converge to the "closest minimum" for certain values or intervals of **w** lead to the question of whether these intervals could be calculated. If so, then the algorithm was assured of converging to the "closest minimum" when it was within an interval of convergence. If the value of **w** was outside the interval, then perhaps a different algorithm (one which did not cause divergence to some extreme point) could be applied temporarily until entering an interval of convergence.

To find the interval of convergence for the second order back-prop, we first find the interval of convergence for the general second order expression.

Let

$$G(w) = w + S(w) \tag{30}$$

where

$$S(w) = - \frac{\frac{\partial E_p}{\partial w}(w)}{\frac{\partial^2 E_p}{\partial w^2}(w)} \quad .$$

We will call **S(w)** the stepping function since it describes how much to increase or decrease the weight **w** with each iteration. The sequence

$$G(w_{n+1}) = G(w_n)$$

is guaranteed to converge if

$$-1 < G'(w) < 1 \quad .$$

Now,

$$G'(w) = 1 + S'(w)$$

so the sequence **G(w)** will converge if

$$-2 < S'(w) < 0 \quad .$$

We will define **S'(w)** to be the convergence factor.

For the general second order expression, the convergence factor is

(31)

$$S'(w) = \frac{\frac{\partial E_p}{\partial w}(w)\,\frac{\partial^3 E_p}{\partial w^3}(w)}{\left(\frac{\partial^2 E_p}{\partial w^2}(w)\right)^2} - 1$$

or

(32)

$$S'(w) = -S(w)\,\frac{\frac{\partial^3 E_p}{\partial w^3}(w)}{\frac{\partial E_p^2}{\partial w^2}(w)} - 1$$

where we approximate the third partial derivative

$$\frac{\partial^3 E_p}{\partial w^3}(w) \approx \frac{\Delta \frac{\partial^2 E_p}{\partial w^2}(w)}{\Delta w} \quad .$$

This expression (32) is also valid for the second order training algorithm

(33)

$$W_1 = W_0 - \frac{\frac{\partial E_p}{\partial W}(W_0)}{\left|\frac{\partial^2 E_p}{\partial W^2}(W_0)\right|}$$

converges to a local minimum provided

$$\frac{\partial^2 E_p}{\partial W_0^2} > 0 \quad .$$

When the proviso is not true, the algorithm can be expected to diverge, as is the case when $\mathbf{W_0}$ is near a local maximum. As we will show shortly, diverging away from a local maximum does not necessarily mean the algorithm is converging to a minimum. This depends upon the nature of the function such as the step size and distance from the current point **w** to a region of convergence near a minimum.

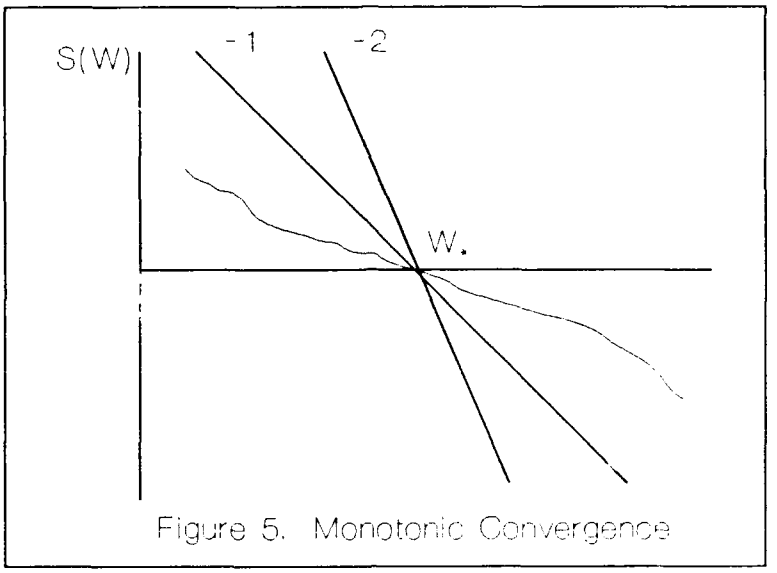


Figure 5. Monotonic Convergence

The above expression (33) is only a sufficient condition for convergence. Figure 5 shows a stepping function, **S(w)**, which meets this condition. Its derivative always remains between 0 and -2. Hence, the sequence **G(w)** is guaranteed to converge to w_*.

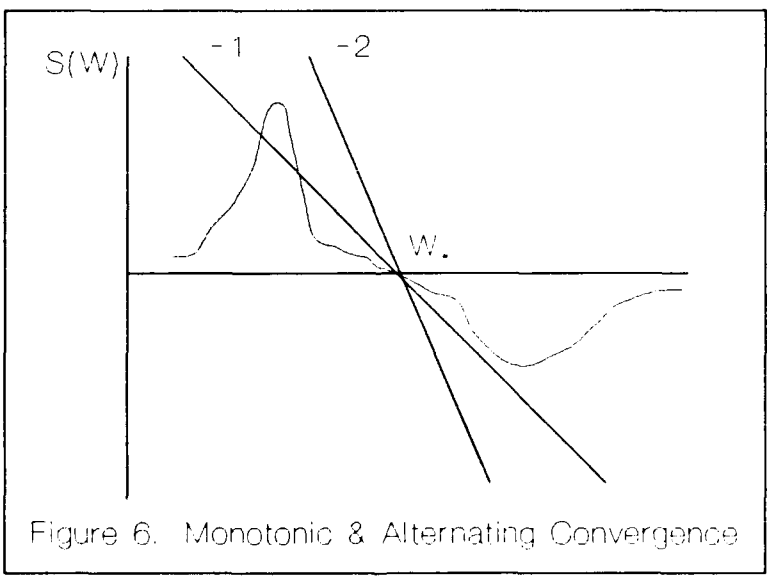


Figure 6. Monotonic & Alternating Convergence

Figure 6, however, illustrates a stepping function which does not meet the convergence condition, but, nevertheless, still converges to point $\mathbf{w}_*$. Notice that so long as **S(w)** remains situated between the line **S = 0** and the line passing through $\mathbf{w}_*$ with a slope of -2, then the sequence **G(w)** will converge to $\mathbf{w}_*$. Notice too, if **S(w)** remains between the lines **S = 0** and the line passing through $\mathbf{w}_*$ with slope of -1, then the sequence **G(w)** will converge monotonically to point $\mathbf{w}_*$. Should **S(w)** fall between the lines having slopes of -1 and -2 and passing through $\mathbf{w}_*$, then the sequence **G(w)** will converge to $\mathbf{w}_*$ in an alternating sequence. (These two lines with slopes of -1 and -2 are shown in figure 6.) The rate of convergence depends upon how close **S(w)** follows the line with a slope of -1. The closer **S(w)** is to this line, the faster the rate of convergence. When **S(w)** is directly on this line, then the sequence **G(w)** will converge to point $\mathbf{w}_*$ in just one iteration. For this example in figure 6, it can be easily seen that the rate of convergence of the sequence **G(w)** varies considerably. In summary, so long as **S(w)**, is between the lines with slopes 0 and -2 the sequence **G(w)** will converge.

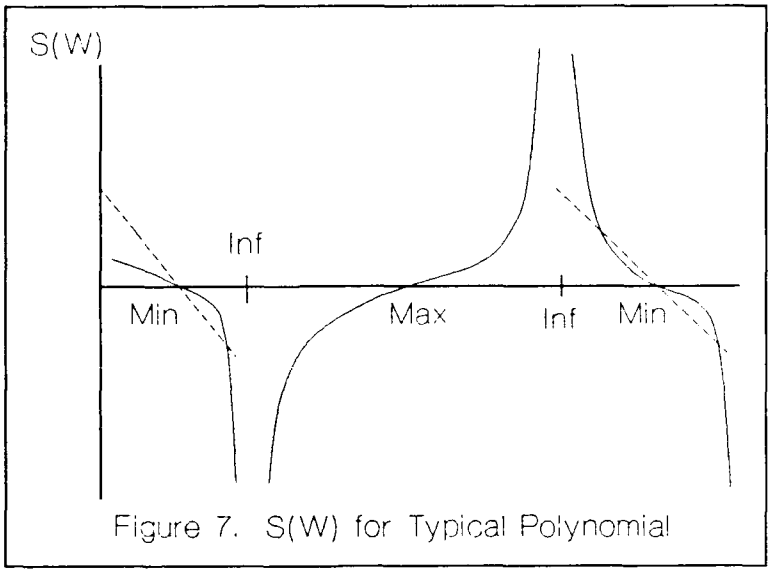


Figure 7. S(W) for Typical Polynomial

Figure 7 shows **S(w)** for some polynomial. Observe how **S(w)** extends outside the region of monotonic convergence defined by **S(w)** = 0 and line(s) whose slope is -1. This occurs whenever the function **S(w)** is undefined or in the neighborhood of an inflection point. Also, note the value of **S(w)** is zero at all minimum and maximum points. However, for certain intervals of convergence it can be seen that **S(w)** is converging quite slowly (i.e., not very near to the line whose slope is -1). This is most pronounced at the extremities of the given polynomial. It is also quite possible that this may be the same situation for an error function. If so, its rate of convergence could be very slow for very large values of **w**.

It appeared that this problem might be solved by increasing the step size, **S(w)**, beyond the calculated value. If successful, then the algorithm could converge to a minimum much faster. The term used to describe this increase is hereby referred to as an over-acceleration term. However, one must be careful when over-accelerating the convergence of an algorithm. If the increase is excessive, then the algorithm will over-shoot the minimum and may fail to converge. This could be especially disastrous since we desire the training algorithm to find the "closest minimum" to some point **w** in the neighborhood of zero. Since **S(w)** generally indicates the closeness to a local minimum, it can be used to inform us when to over-accelerate. Therefore, over-acceleration will be used only when **S(w)** is large (i.e., greater than 10), and then only if the result causes the function to decrease in value. (If **S(w)** is large, we can assume the minimum is far away.) Over-acceleration will not be used whenever **w** is near zero, regardless of the value of **S(w)**. It was decided to use the following expression to increase the step size.

(34)

$$S(w_1) = \begin{cases} S(w_0)\ \log_{10}|S(w_0)|; & |S(w_0)| > 10 \\ S(w_0) \qquad\qquad\quad ; & |S(w_0)| \leq 10 \end{cases}$$

where

$S(w_1)$ *is the step update*
$S(w_0)$ *is the current step*

This expression was chosen because it generates a small increase in step size when the calculated step size is small, and a large increase in step size when the calculated step size is large. The principal advantage of over-acceleration is, when successful, it can save a number of iterations. However, it does require a small amount of additional time to calculate and test - a small price to pay considering the potential savings.

Over-acceleration was incorporated into the second order algorithm and applied to the given polynomial with very good results. When the weight, **w**, was initialized far from a minimum, over-acceleration reduced the number of iterations by factors as high as three.

Now, that the convergence factor **S'(w)** could be calculated and used to tell whether the general second order algorithm was within a region of convergence (i.e., converging to the closest minimum), the next problem was how to best increase the number of weights with values in the neighborhood of zero, thereby increasing the number of candidates for pruning.

The error function could be slightly altered as follows to help achieve this end (Oxley, 1990):

(35)

$$E_p = \sum_{k=1}^{K} \frac{1}{2}\left(D_k - f_k\left(\sum_{j=1}^{J} W2_{j,k} f_j\left(\sum_{i=1}^{I} W1_{i,j} X1_i\right)\right)\right)^2 + \frac{1}{N}\left(\vec{W1}^T \vec{W1} + \vec{W2}^T \vec{W2}\right)$$

where N is the iteration count.

The additional term added to the original error function is a weight adjustment term. With the inclusion of the weight adjustment term, the weights will be minimized along with the original error function. Using the factor 1/N, will ensure the weight adjustment term has less and less effect as the iteration count increases. For large N the term is essentially zero leaving the original error expression.

In addition, including the weight adjustment term causes the first and second partial derivatives to change to the following:

$$\frac{\partial E_p}{\partial W2_{j,k}} = (Y2_k - D_k)\ f'_k(S2_k)\ Y1_j + \frac{2}{N} W2_{j,k}$$

$$\frac{\partial^2 E_p}{\partial W2_{j,k}^2} = Y1_j^2\ f'_k(S2_k)^2 + Y1_j^2\ f''_k(S2_k)\ (Y2_k - D_k) + \frac{2}{N}$$

$$\frac{\partial E_p}{\partial W1_{i,j}} = \sum_{k=1}^{K} (Y2_k - D_k)\, f'_k(S2_k)\, W2_{j,k} f'_j(S1_j)\, X1_i + \frac{2}{N} W1_{i,j}$$

$$\frac{\partial^2 E_p}{\partial W1_{i,j}^2} = \sum_{k=1}^{K} f'_k(S2_k)^2\ f'_j(S1_j)^2\ W2_{j,k}^2\ X1_i^2$$
$$+ (Y2_k - D_k)\, f''_k(S2_k)\, f'_j(S1_j)^2\ W2_{j,k}^2\ X1_i^2$$
$$+ (Y2_k - D_k)\, f'_k(S2_k)\, f''_j(S1_j)\ W2_{j,k}\ X1_i^2$$
$$+ \frac{2}{N} \quad .$$

To test the effectiveness of the algorithm which includes the weight adjustment term on finding weights with values close to zero, the modified error expression was applied to the polynomial. The results showed the algorithm consistently found the minimum nearest to the origin.

However, concern still remained about how the algorithm with weight adjustment would fair when encountering a point of inflection. Should this occur when N was very large, it is doubtful that the weight adjustment term would be able to have any effect towards "pulling" the weights back towards the origin. Also, another concern still existed. Should the weights be in the neighborhood of a maximum, divergence may cause the algorithm to overshoot the nearest minimum. These problems still required further study of the convergence properties of the second order algorithm.

Now, when the absolute value sign was imposed on the denominator of Newton's second derivative expression, it changed the algorithm such that it would only enter a region of convergence when nearing a local minimum. It will diverge elsewhere. The underlining question here is, can the algorithm be made to diverge in a graceful manner when it is near a maximum point or point of inflection? In other words, is it possible to control the rate of divergence away from these points so that the algorithm is migrating in a well behaved manner toward the "closest minimum"? If so, how do we control it, and for what value of **w** do we impose the control? If that value could be estimated, then perhaps a different algorithm could be temporarily employed until it enters a region of convergence defined by **S'(w)**. Once this region is

entered, then the second order algorithm could be re-employed until convergence is achieved to the minimum point.

To answer these questions, we examine three cases of possible minimum/inflection/maximum conditions. The first of these is the case where the inflection point is midway between the minimum and maximum points. This is illustrated in figure 8.

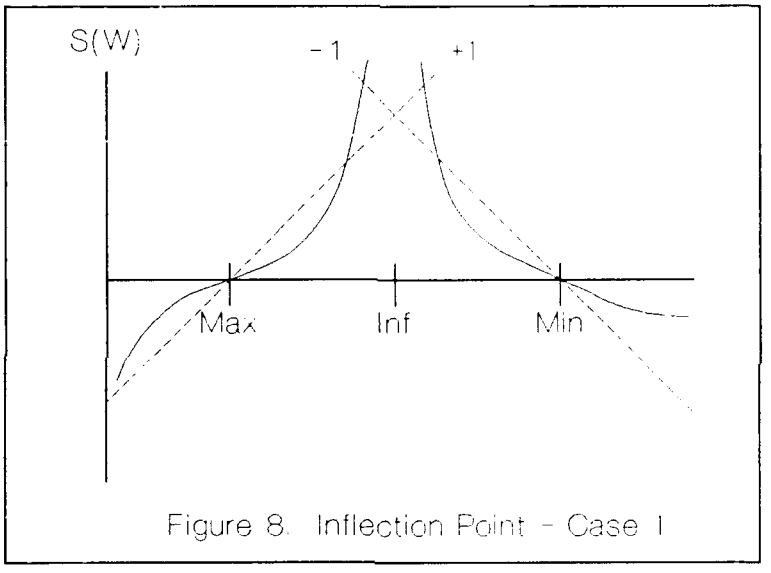


Figure 8. Inflection Point - Case 1

When the algorithm is near the maximum point, it diverges away in the direction of the minimum. It continues to diverge with each iteration until it reaches some value of **w** where it is no longer monotonically approaching the "closest minimum". In figure 8, this point is where **S(w)** is above the line whose slope is +1. Therefore, if the algorithm is controlled by allowing it not to continue once **S'(w)** = 1, then the algorithm will not have overshot

the minimum point. By using either the previous step size or a step size based upon the desired resolution, one could iterate past the inflection region into the interval of convergence. (Note: Desired resolution is the minimum distance between two adjacent minima one hopes to resolve.) Once **S'(x)** is between 0 and -1, Newton's algorithm could be re-engaged and allowed to converge to the minimum point.

Figure 9 illustrates the second case. This situation shows the inflection point lying closer to the maximum point than the minimum. Again, by choosing **w** where **S'(w) =** -1, it can be seen that the step size will not cause the algorithm to overshoot the minimum point.

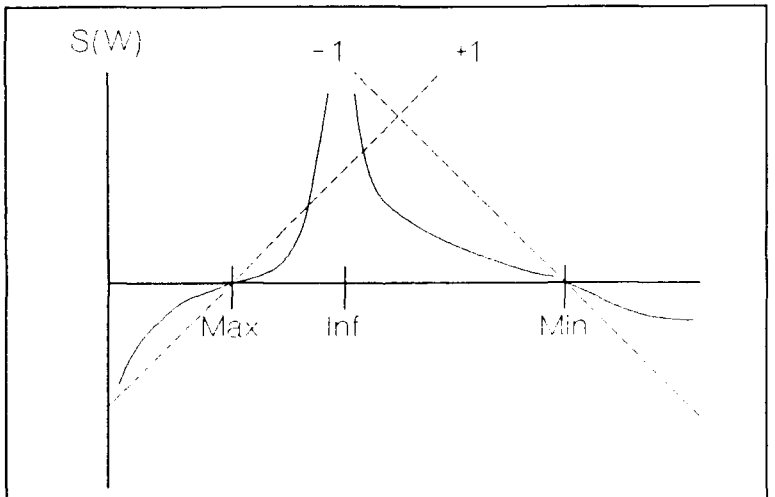


Figure 9. Inflection Point - Case II

Like the first case, once **S'(w) > 1**, either the previous step size or a step size based upon the desired resolution could be instituted. This step size could be maintained while transversing through the inflection region until the convergence factor, **S'(w)**, has a value between 0 and -1. When this occurs, then Newton's second order algorithm could be re-employed until convergence is achieved.

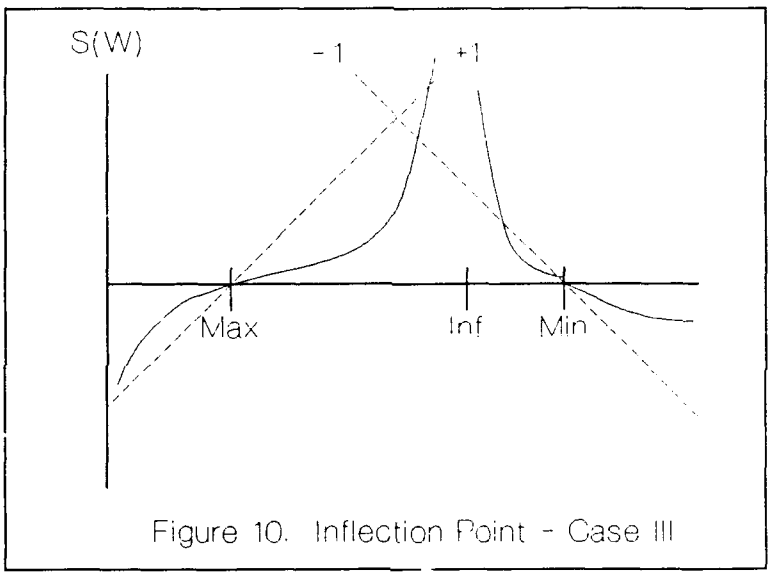


Figure 10. Inflection Point - Case III

The third and last case is shown in figure 10. In this instance the inflection point is closer to the minimum. The situation here is different than the other cases because choosing the value of **w** where the convergence factor **S'(w) = 1 may** now cause the algorithm to overshoot the "closest minimum". Furthermore, this could occur for any **w** where **S'(w) < 1**. The closer **S(w)**

follows the line whose slope is +1, the quicker the algorithm will diverge and the more likely it will overshoot. The converse is also true. This can be readily seen if one visualizes possible intervals where **S'(w) = 1** for different **S(w)**. (Note: **S(w)** must be monotonically increasing on the interval bounded by the maximum point and the point of inflection.) In addition, the nearer the inflection point is to the minimum, the more likely the algorithm will overshoot it. Unfortunately, there is no value of **S(w)** that will guarantee convergence to the closest minimum for every error function under these circumstances. One possible solution would be to use a step size based upon the desired resolution during the entire interval when the error function is in a region of a maximum or point of inflection. However, this "brute force" method would be quite slow especially when the desired resolution is small. The best alternative appears to be to the same as outlined above for the other two cases. That is, use Newton's algorithm until the convergence factor **S'(w)** is greater than one, then use either the previous step size or one based upon the desired resolution. The step size would be held constant at this value until reaching an interval of convergence where the convergence factor, **S'(w)**, is between 0 and -1. This is perhaps the best we can hope for under this situation. Many error functions, in particular those with slow rates of divergence, may very well indeed converge to the "closest minimum". But again it's important to emphasize the

convergence is not guaranteed. Much depends upon the nature of the error function and the relative distances of the critical points.

Recapping, to increase the chances of convergence to the "closest minimum", Newton's second order algorithm will be used so long as the magnitude of the convergence factor, $S'(w) < 1$. When the convergence factor is greater than or equal to one, the previous step size or one based upon the desired resolution will be used.

To test this algorithm, it was applied to the polynomial to observe how well it converged to the closest minimum for various starting values. (The weight adjustment term was not used during this test.) The results showed that the algorithm consistently converged to the "closest minimum" even when the initial starting value was near a maximum or inflection point. However, it must be pointed out that the ability of the algorithm to consistently converge to the "closest minimum" for any starting value of w, may be due to the nature of the polynomial. Other polynomials may not necessarily give the same results depending upon the positions of the inflection points relative to the minimum and maximum points.

Error Predicting Function

Now that the algorithm is formulated that will minimize the pattern error having weights close to zero, the one last item which will be derived is the error predicting function. This function will be used to predict the change in pattern error when a weight is removed (nulled). Beginning with a few terms of the Taylor series:

(36)

$$E_p(\vec{W}_1) \approx E_p(\vec{W}_0) + \vec{S}^T \nabla E_p(\vec{W}_0) + \frac{1}{2}\vec{S}^T \nabla^2 E_p(\vec{W}_0)\vec{S}$$

where

$$\vec{S} = \vec{W}_1 - \vec{W}_0 \quad .$$

Now, let

$$\Delta E_p = E_p(W_1) - E_p(W_0)$$

then (37)

$$\Delta E_p = \vec{S}^T \nabla E_p(\vec{W}_0) + \frac{1}{2}\vec{S}^T \nabla^2 E_p(\vec{W}_0)\vec{S} \quad .$$

Notice that the gradient will be approximately zero at a local minimum. Therefore, the expression can be simplified to:

(38)

$$\Delta E_p = \frac{1}{2} \vec{S}^T \nabla^2 E_p(\vec{W}_0) \vec{S} \quad .$$

Ignoring the off diagonal terms of the Hessian matrix, the error predicting function can be further simplified to:

(39)

$$\Delta E_p = \frac{1}{2} \sum_n \frac{\partial^2 E_p}{\partial W^2} (W_{0(n)}) S^2_{(n)}$$

where

$$S_{(n)} = W_{1(n)} - W_{0(n)} \quad .$$

When only one weight is nulled, the expression becomes

(40)

$$\Delta E_p = \frac{1}{2} \frac{\partial^2 E_p}{\partial W^2} (W_0) S^2$$

where

$$S = W_1 - W_0 \quad .$$

Network Reduction

Various size neural networks were constructed to test the net reduction technique on the XOR problem. The first net constructed was comprised of a 10 hidden nodes. The output node was configured for a sigmoid output. The net was trained to a total error of 0.01 or less over all training samples. When the net was first trained using the first order algorithm, a total 4 weights were found to have values between -1 and 1. Next, the second order algorithm with relaxation was employed. Not only was the weight adjustment term included, but the program included a check of the convergence factor. When the convergence factor, **S'(w)**, had a magnitude of 1 or more, then the previous step size was used until the magnitude of **S'(w)** once again fell between 1 and 0. Should a previous step size not yet exist (i.e., first iteration), then a step size of 0.1 was used. As a result, the second order algorithm was able to train the net, and arrive with 7 weights having values between -1 and 1. This provided an improvement of three additional weights over the first order algorithm. The seven weights and their values are listed below in table 2. Next to each weight is the predicted error upon the removal of the weight. The column on the far right is the error measured when a test pattern was applied to the network with the weight removed.

Table 2.
Pruning Error (10 Hidden Nodes)

Weight	Value	Pred. Error	Meas. Error
1	-0.82317	3.40×10^{-3}	4.20×10^{-4}
2	-0.46541	-6.09×10^{-6}	1.23×10^{-6}
3	0.99105	2.01×10^{-5}	4.88×10^{-5}
4	-0.29116	1.94×10^{-7}	7.45×10^{-6}
5	0.51534	5.63×10^{-5}	2.11×10^{-5}
6	0.38774	8.90×10^{-5}	3.50×10^{-5}
7	-0.99846	4.52×10^{-4}	1.99×10^{-5}

In this network, of the 7 weights, no 2 weights were found to be inputs associated with the same node. Therefore, no node was removed. All of the weights except the first were removed from the net, and the network re-trained - this time using only the second order algorithm. After the net was trained only 3 weights were found with values near zero. Their predicted and measured errors were small (i.e., < 0.00001) so they were pruned. Two of these weights were associated with the same node, and so the node was removed too. Again, the net was retrained; however, this time the net was only able to achieve an error of 0.17. Since this did not meet the intended goal of 0.01, the process was terminated.

For the second test, a network was constructed with 15 nodes in the hidden layer. Like the first net, this net was also configured for a sigmoid output, and was trained to a total error of 0.01 or less over all training samples. When the first order algorithm was used to train the net, a total of 9 weights were found to have values between -1 and 1. When the second order algorithm was used, 14 weights were found to have values between -1 and 1. These weights are listed below in table 3 along with their predicted and measured errors upon removal.

Table 3.
Pruning Error (15 Hidden Nodes)

Weight	Value	Pred. Error	Meas. Error
1	-0.056235	2.6×10^{-7}	7.3×10^{-6}
2	0.869313	9.3×10^{-6}	9.8×10^{-6}
3	0.554017	2.3×10^{-6}	3.4×10^{-7}
4	-0.673478	4.1×10^{-5}	8.3×10^{-5}
5	0.311442	7.3×10^{-5}	6.6×10^{-5}
6	0.769293	8.3×10^{-6}	9.8×10^{-7}
7	0.662095	8.5×10^{-5}	7.5×10^{-5}
8	-0.038621	6.3×10^{-7}	2.0×10^{-6}
9	0.099876	9.1×10^{-7}	5.5×10^{-7}
10	-0.372197	6.3×10^{-6}	1.3×10^{-5}
11	-0.432888	4.3×10^{-5}	7.7×10^{-6}
12	0.795237	7.7×10^{-6}	8.4×10^{-6}
13	0.598286	8.3×10^{-7}	-4.3×10^{-5}
14	-0.542693	1.4×10^{-6}	1.1×10^{-5}

These weights all had small errors associated with them, so they were all candidates for removal. Of the 14 weights, no two weights were found to be associated with a single node; therefore no nodes were pruned. Once the weights were removed, the net was retrained. However, after only one pruning process this net, like the previous net, failed to achieve the desired error of 0.01. It was only able to achieve a total error of 0.13, so pruning ended.

IV Summary and Conclusion

This research accomplished its major objective, namely to reduce a network by eliminating weights using error prediction. However, the technique was not able to reduce a network down to its theoretical minimum size. This minimum size for the XOR problem is three nodes. For networks which underwent reduction, each failed to reach the minimum size due to excess error after pruning. The networks were not able to achieve an error of less than 0.01 after retraining with weights removed. It is difficult to provide a reason for this, but it may be due to the manner in which the algorithm drove itself to find a local minimum with weights close to zero. It is quite possible that other local minima exist which may not have weights of low value, but nevertheless do provide lower values of error. It is difficult to draw a definite conclusion not having a complete picture of the error surface.

Another technique which was not all that successful was predicting the amount of error associated with a weight when it was removed from the net. For some examples the predicted error came quite close to the actual error, but for others it was off by several magnitudes. For a couple instances, the predicted error was the wrong sign. The reason for these differences could be the fact that the numbers used to calculate the error were so small

that round off error could be causing loss of precision. Also surprising, was the difficulty involved in choosing which weights from among those close to zero should be removed. It was originally hoped that this decision process would be made easy by having some weights with very small errors associated with them, and other weights having very large errors. However, as the results show, it was difficult to choose which weights to remove due to the way in which the error was distributed among the weights. One conclusion which the results do seem to support is that the error associated with weights, for the most part, is distributed somewhat evenly. This may be the reason why a network tends to degrade in a graceful manner when some of its nodes are eliminated. Originally, it was hoped to use the full Hessian matrix during training and for error prediction; however, this proved to be too costly in terms of computational requirements so the idea was dropped in favor of Le Cun's diagonal method. Perhaps this approximation is the cause of some of the error, but it is not believed to be a major contributor, otherwise the algorithm may not have been able to train the net.

On the other hand, this research did have a number of successes and a few surprises. First, the process by which a minimum was found having weights close to zero did work quite well - first with the polynomial, then with the network. In both cases the number of weights with values close to zero increased when the algorithm which included the weight adjustment term was employed. Also, the process by which the algorithm checked the value of **S'(w)** to determine if it was within an interval of convergence performed satisfactorily - at least on the polynomial. Along with checking **S'(w)**, the technique of using the previous step size (or a step sized based upon the desired resolution) was also quite successful on the polynomial at controlling the rate of divergence when passing through an inflection point. It is difficult to draw a definite conclusion for the network, since the error surface is unknown. Nevertheless, the results appear to show that the process worked well for the network too, since the number of weights near zero increased in number. This suggests that regions of divergence were either not encountered, or were controlled so the algorithm would not diverge to some far distant set of weights.

One surprise finding was how well the first order algorithm converged when using a variable eta. Although it did not converge in less iterations than the second order algorithm, it did do much better than the first order algorithm using a fixed eta. It also performed better than the first order algorithm which used a momentum term.

Another surprise finding was that both the predicted and the actual errors increased when a weight was removed. It might be expected to have a handful of weights which would cause the error to decrease. However, this was not the case. In almost every example, removing the weight caused the error to increase. One explanation for this may be the algorithm's effectiveness to locate a local minimum. This would explain the increase in error when a weight is perturbated.

Final Recommendations

The results presented here suggest that reducing a network using error prediction may not to be that effective or reliable. As discussed above, the estimated error was not very close to the actual error one received after removing the weight. Also, the distribution of the predicted error among the weights made it difficult to determine which weights to eliminate. Furthermore, once the network was pruned a certain amount, it was unable to be trained to the desired error.

In addition, this research suggest using a varying eta with the first order algorithm may increase the rate of convergence over an algorithm which uses a constant eta. Additional testing on other networks will be necessary before any overall conclusions can be reached, but the initial results look promising. However, where a choice exists between using a first order training algorithm versus a second order algorithm, the second order algorithm should be chosen because of its improved convergence properties.

Bibliography

1. Ahmad, Subutai. "Scaling and Generalization in Neural Networks: A Case Study." In Touretzky, David S., editor, Advances in Neural Information Processing Systems 1, pages 160-168, San Mateo, CA: Morgan Kaufmann, 1989.

2. Arbenz, Kurt and Wohlhauser, Alfred, Advanced Mathematics for Practicing Engineers", Artech House, 1986.

3. Baba, Norio. "New Approach for Finding the Global Minimum of Error Function of Neural Networks." Neural Networks Vol 2, No. 5, pages 367-373, 1989.

4. Baum, Eric and David Haussler. "What Size Net Gives Valid Generalization?" In Advances in Neural Information Processing Systems I, pages 81-90, San Mateo, CA: Morgan Kaufmann, November 1989.

5. Chiueh, T.D. and Goodman, R. M. "Learning Algorithms for Neural Networks with Ternary Weights." Neural Networks, Vol 1, No. 1 Suppl, page 166, 1988.

6. Cybenko, G. "Approximations by Superpositions of Sigmoidal Functions," Matheatics of Control, Signals, and Systems (1989). Accepted for publication.

7. Dennis, J. E., Jr. and Schnabel, Robert B., "Numerical Methods for Unconstrained Optimization and Non-Linear Equations", Prentice-Hall, 1983.

8. Ishikawa, Masumi. "A structured Learning Algorithm with Forgetting of Link Weights" International Joint Conference on Neural Networks, Washington DC 1989.

9. Le Cun, Y. and Becker, S. "Improving the Convergence of Back-Propagation Learning with Second Order Methods," Proceedings of the 1988 Connectionists Models Summer School, D. Touretzky, G. Hinton, and T. Sejnowski, eds., pages 29-37, San Mateo, CA: Morgan Kauffman, 1989.

10. Le Cun, Yann; Denker; and Solla. "Optical Brain Damage." In Touretzky, David S., editor, Advances in Neural Information Processing Systems 2, pages 598-605, San Mateo, CA: Morgan Kaufmann, 1990.

11. Oxley, Mark, Professor. Notes and discussion. Dept of Mathematics and Computer Science, Air Force Institute of Technology (AU), Wright Patterson AFB, OH 1990.

12. Rogers, Steven Maj, Phd Class notes. Dept of Engineering, Air Force Institute of Technology (AU), Wright Patterson AFB, OH 1990.

13. Ruck, Dennis W., et al. "Back Propagation: A Degenerate Kalman Filter?, "IEEE Transactions on Pattern Analysis and Machine Intelligence (June 1989). Submitted for publication.

14. Ruck, Dennis W., et al. "Feature Selection Using a Multilayer Perceptron," Journal of Neural Network Computing (1990). Submitted for publication.

15. Tarr, Gregory L., "Dynamic Analysis of Feedforward Neural Networks using Simulated and Measured Data. MS thesis, AFIT/GE/ENG/88D-54 Air Force Indtitute of Technology (AU), December 1986 (AD-A177598).

16. Sietsma, Jocelyn. "The Effect of Pruning a Back-Propagation Network" DSTO Materials Research Laboratory, Ascot Vale, Victoria, Australia 3032, 1990.

17. Singhal, Sharad and Wu, Lance. "Training Multilayer Perceptrons with the Extended Kalman Algorithm." In Touretzky, David S., editor, Advances in Neural Information Processing Systems 1, pages 133-140, San Mateo, CA: Morgan Kaufmann, 1989.

18. Wang, Shengrui. "Training Multi-Layered Neural Networks with T.R. Based Algorithm." Neural Networks Vol 1, No. 5 Suppl, page 228, 1988.

19. Werbos, Paul J. Beyond Regression: New Toools for prediction and Analysis in Behavior Sciences. Phd dissertation, Harvard University, Cambridge, Mas., 1974.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	4 DEC 1990	Master's Thesis

4. TITLE AND SUBTITLE

NETWORK REDUCTION USING ERROR PREDICTION

5. FUNDING NUMBERS

6. AUTHOR(S)

Michael V. Gilsdorf, Captain, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Air Force Institute of Technology, WPAFB OH 45433-6583	AFIT/GE/ENG/90D-24

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Lt Altquist Rome Air Development Center Intelligence and Reconnaissance Div IRRA 587-3175	

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
Approved for public release; distribution unlimited	

13. ABSTRACT (Maximum 200 words)

This thesis investigates gradient descent learning algorithms for multi-layer feed forward neural networks. A technique is developed which uses error prediction to reduce the number of weights/nodes in a network. The research begins by studying the first and second order back-prop training algorithms along with their convergence properties. A network is reduced by making an estimate of the amount of error which would occur when a weight(s) is removed. This error estimate is then used to determine if a particular weight is essential to the operation of the network. If not, it is removed and the network retrained. The process is repeated until the network is reduced to the desired size, or the error becomes unacceptable.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Network Reduction, Network Pruning, Convergence, Second Order Back Propagation	80
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines to meet optical scanning requirements.**

Block 1. Agency Use Only (Leave Blank)

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C	- Contract	**PR**	- Project
G	- Grant	**TA**	- Task
PE	- Program Element	**WU**	- Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency. Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ..., To be published in .... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availablity Statement. Denote public availability or limitation. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR)

DOD	- See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE	- See authorities
NASA	- See Handbook NHB 2200.2.
NTIS	- Leave blank.

Block 12b. Distribution Code.

DOD	- DOD - Leave blank
DOE	- DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports
NASA	- NASA - Leave blank
NTIS	- NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.